Proving process calculi translations in ecrins :The pureLOTOS -> MEIJE example

Guillaume Doumenc, Eric Madelaine, Robert de Simone

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél (1) 39 63 55 11

Rapports de Recherche

N° 1192

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

PROVING PROCESS CALCULI TRANSLATIONS IN ECRINS : THE PURELOTOS ---> MEIJE EXAMPLE.

Guillaume DOUMENC
Eric MADELAINE
Robert DE SIMONE

Mars 1990

RR-1192

Proving process calculi translations in ECRINS : The PureLOTOS $\mapsto$ MEIJE example.

Preuves de traductions entre calculs de processus dans ECRINS : L'exemple PureLOTOS $\mapsto$ MEIJE.

Guillaume Doumenc [1]
Eric Madelaine
Robert De Simone

I.N.R.I.A.
Route des Lucioles, Sophia Antipolis
06 565 VALBONNE Cedex (FRANCE)

gdo@mirsa.inria.fr
madelain@mirsa.inria.fr
rs@cma.cma.fr

Résumé Le système ECRINS permet la manipulation symbolique dans les algèbres de processus. Un calcul de processus est defini par la donnée de la syntaxe et des règles de comportement opérationnelles de ses opérateurs. De cette définition, le système déduit un algorithme d'évaluation des comportements des termes de l'algèbre. Cet algorithme est symbolique, c'est à dire qu'il permet de traiter des expressions comportant des variables libres de processus; les comportements de telles expressions sont décrits par un ensemble de règles conditionnelles. Le système permet également de prouver la validité de lois équationnelles entre expressions d'une algèbre de processus. L'équivalence utilisée est la bisimulation forte. Ce type de preuve est utilisé aussi pour vérifier la validité de traductions d'un calcul de processus dans un autre. Nous donnons l'exemple d'une traduction d'un sous-ensemble de LOTOS dans MEIJE-SCCS, et donnons quelques fragments de la preuve de sa validité.

Abstract The ECRINS system is a tool for symbolic manipulation in process algebras. A process calculus is defined by the syntax and the conditional behaviour rules of its operators. From such a definition, the system builds an algorithm for evaluating the behaviours of the algebra terms. This computation is symbolic, and it can manage expressions with free process variables: the behaviours of such expressions are described by conditional operational rules. Another activity of ECRINS is the proof of equational laws w.r.t. strong bisimulation. Such proofs are used for proving correct intercalculi translations. We give the example of a translation from a subset of the LOTOS language into the MEIJE-SCCS algebra, with some pieces of the validity proof.

[1] First author's contribution was carried out as part of a thesis at the IBM *Centre Etudes et Recherches* in La Gaude (FRANCE).

Proving process calculi translations in ECRINS : The PureLOTOS $\mapsto$ MEIJE example.

Guillaume Doumenc
Eric Madelaine
Robert De Simone

I.N.R.I.A.
Route des Lucioles, Sophia Antipolis
06 565 VALBONNE Cedex (FRANCE)

gdo@mirsa.inria.fr
madelain@mirsa.inria.fr
rs@cma.cma.fr

1 Introduction.

Process calculi theory was started as a mean to provide strong syntactic and semantic foundations to the study of parallelism and concurrent communication systems. This made it possible to design tools which would directly work on the semantical apparatus (here Plotkin's Structured Operational Semantics rules) as first-class objects. Such tool would use the algebraic structure of terms to conduct activities running from plain behaviours evaluation, to checking of process expressions equivalences of some sort, and ultimately to different calculi comparisons via embeddings and translations correctness proofs.

We present here the ECRINS software system, which is a first attempt at defining such a tool. We examplify its functional use by running an automatic proof of a human-defined translation from a pure version of the LOTOS language into the MEIJE algebra. Here pure means without variables and value-passing. MEIJE is a language which owes most of its operators to CCS, and in addition incorporate the latter SCCS simultaneity product, with an asynchronous parallel mechanism [Bo85]. The validity proof is structural: each operator of PureLOTOS is proved to be in strong bisimulation with its corresponding MEIJE expression.

2 Process calculi.

Process algebras

Process calculi are now a well-accepted generic notion for designing a class of formalisms which share the same definitional principles : CCS [Mi80], SCCS [Mi83], MEIJE [Bo85], TCSP [Br83], ACP [BK85], PureLOTOS [Ai86] to name a few. We shall assume reader's acquaintance with at least one of these languages and its definitional mechanisms.

Process calculi are based on two main types : *actions* and *processes*. Operators take actions and processes arguments into processes, providing an classical algebraic structure.

Operational semantics provides interpretation of closed terms into transitions systems, with actions as transition labels. Operators and non-closed expressions are then interpreted as transition system transformers; this semantics is defined through a specific class of Plotkin's style "Structural Operational Semantics" conditional rewrite rules [Pl81]. We shall describe further this class while presenting ECRINS in the next section; see also [DSi85].

Special operators are action renamings and recursive definitions. They are present in all process calculi, and as such are treated generally by ECRINS.

Actions

In all process algebras existing so far, actions are themselves structured. This structure is what allows for synchronization and further communication to be handled in relevant operators rules. Just recall the inverse signals in CCS, which meets in synchronization and produce a hidden action τ. In addition in SCCS and MEIJE there is a full commutative monoid of potential simultaneous actions, again containing a group of invertible signals. Actions structure in ACP is more loose and parametric, while in PureLOTOS it is only a set of so-called gate names without structure, but for a distinguished termination action δ.

Some operators in these process algebras have a neat scoping role with respect to actions encapsulation (restriction in CCS, SCCS, MEIJE, ACP and hiding in ACP, PureLOTOS). As ECRINS will deal in full generality with calculi described only by abstract acceptable rules for operators, this scoping function will need be made explicit with the further construct :

local <*action*> **in** <*process expression*>

This construct resembles formally the λ-abstraction of λ-calculus : not only is the action name declared to be local to the given expression, but also instanciation of any process variable appearing in it will lead to α-conversion to avoid name clash.

3 Ecrins

ECRINS is a manipulation system for process calculi, process calculi semantics and process calculi expressions. It can be split in three main parts : Compiling a calculus description to get a parser for terms. evaluating the behaviours of expressions and proving bisimulation equivalences.

3.1 Compiling and Parsing

A process calculus must be described in a calculus description file. This file is to be analysed by the *calculus compiler*. Then terms of the calculus can be parse and manipulated by ECRINS. The calculus description file contains the syntactic definition of the calculus operators; the *calculus compiler* uses this part to produce a scanner and a parser for expressions of the calculus.

```
@ parse x = local signal s in (p[s!/a] // q[s?/a])\s!;
x : Process of meije
```

In addition one has to provide semantics for such operators, using a specific syntax for structural operational rules. This allowed syntax is describe below. Theses rules in

themselves are the main object manipulated inside ECRINS. They are parsed and analysed for coherency at this level.

We now sketch the allowed syntax formats using examples. Full documentation will be found in [MaDSiVe89] and is recalled shortly below.

```
operator hiding:: Process Label --> Process
syntax
       \\   left 4
semantics
   hiding                  p -- a --> p'  &  (a equal s)
                           ----------------------------
                              p\\s -- tau --> p'\\s

   hiding-not
                         p -- a --> p'  &  not (a equal s)
                         --------------------------------
                               p\\s -- a --> p'\\s
end
```

The first line declares the name and the signature of the operator. Then follows the concrete syntax description. along with associativity and priority declarations aimed at the parser generator. The end right upper part of the rules (eg. `(a equal s)`) is a predicate over actions conditioning the applicability of the rule. The plain format of the structural conditional rules will be explain below.

```
operator ticking :: Action Process --> Process
syntax
        * right 3
semantics
        ticking       p -- a --> p'
                      ------------------
                      s*p -- a.s --> s*p'
end
```

```
operator disable :: Process Process --> Process
syntax
        [> left 4
semantics
        disabling_1      p -- a --> p' & not(a equal d)
                         ------------------------------
                             p[>q -- a --> p'[>q

        disabling_2      p -- a --> p' & (a equal d)
                         ---------------------------
                             p[>q -- d --> p'

        disabling_3              q -- a --> q'
                                 ---------------
                                 p[>q -- a --> q'
end
```

A set of one-step evaluation tactics –basically: "Try and apply this rule!"– is compiled from such description. This set may then be used inside a small ML-inspired tactical language (see [MaDSiVe89]), or through a standard strategy that is enougth for the translation problem in this paper.

The allowed Conditional Rewrite Rules format

The rules must obey the following format :

$$\frac{\{x_j \xrightarrow{u_j} x'_j\}_{\mathcal{J}\subset[1..n]} \quad \& \quad P(\{u_j\}_{\mathcal{J}}, a_1, ..., a_m)}{Op(x_1, ..., x_n, a_1, ..., a_m) \xrightarrow{F(\{u_j\}_{\mathcal{J}}, a_1, ..., a_m)} T'(\{x_k\}_{[1..n]-\mathcal{J}}, \{x'_k\}_{\mathcal{J}}, a_1, ..., a_m)}$$

where we call:

- *premisses* the upper part of the rule and *conclusion* its bottom part.
- *subject* the term at the left end part of the conclusion. The head operator Op of the subject has process arguments $x_1, \ldots, x_n$ and action parameters $a_1, \ldots, a_m$. Inside P, F and T' some other actions may appear: they are constants of the calculus and had to be declared as such previously.
- *formal hypothesis* the part of the premisses on the left of the & and *working formal variables* the x_j with $j \in \mathcal{J}$.
- *actions predicate* the part of the premisses on the right of the &. P belongs to boolean operators closure of the following basic predicates : *equality, divisibility, set membership..* over actions terms with synchronization product.
- *resulting action* the label over the arrow of the conclusion. The resulting action F is a function of the formal hypotheses actions (and of the operator action parameters).

- *resulting process* the right end part of the conclusion. The process arguments that appear as formal hypothesis must be transformed as x'_j in the resulting process: you are not allowed to test a potential future behaviour of a process without making it explicitly perform its action within the considered rule, therefore saving the possibility of choosing another future.

3.2 Evaluation

The ECRINS evaluator manipulates tactics inherited from the rules and piles them up in deduction trees to produce one-step behaviours of terms. The basic predicates are transition capabilities of subprocesses of the form $e \xrightarrow{u} e'$; they induce the possible shapes of the trees according to the head operator of e. Then another kind of predicates, this time defining imperative relation to hold in between actions performed by subterms, reduce the domain of validity of this composed subtree. In case no actions whatsoever would fulfill this condition, the subtree should ideally be trimmed out. We shall come back to how ECRINS deals with such predicates further below. But for now we elaborate more about tactics, tacticals and to how to build candidate behaviour trees.

We shall not give an exhaustive list of tacticals used in ECRINS and rather prompt the reader to [MaDSiVe89]. Under certains assumptions on the expression to be evaluated, the most obvious being that there is no unbounded recursive creations of new parallel process, the set of behaviours (or *specification*) is finite. It may then be obtained safely with simple recourse to the aforementioned *All* tactic. This is the most usual case, specially in inter-calculi translations. It is the default option of ECRINS.

```
@ parse x=a:stop+(b:stop//c:stop);
x : Process of meije;

@show eval x;
Specif of: a:stop+(b:stop//c:stop)
  --a--> stop
  --b--> stop//c:stop
  --c--> b:stop//stop
  --b.c--> stop//stop
 : Specif of meije
```

We now turn to predicates upon actions and their inter-relations. Forcing actions performed by the subcomponents and the global expression altogether to keep a certain relation amounts to synchronizing, in a very broad sense. Only certain combinations of actions are allowed. others are discarded. An obvious problem which raises is how to find whenever none is still allowed.

In general, as the possible behaviours of any subexpression may itself be a predicate –and this goes down to eventual process variable at the leaves of the expression–, predicates act as predicates transformers. Then one encounters problems of decidability and efficiency. These predicates could be treated either formally in logical framework, or by taking advantage of the underlying interpretations: either in the finite group case of CCS. or finite monoid case of PureLOTOS, or again in the commutative monoid of SCCS-MEIJE.

ECRINS provides for this three alternative predicate modes of evaluation:

- In *logic* mode, no resolution whatsoever is attempted. The predicates then appeared as formal parts of the specification hypothesis. This mode could be wired to a logical interpreter some days. It has not been yet.

```
Specif of: ((p:stop)\\s)
   {p}{not (s divide u_p)} --u_p--> stop\\s
   {p}{(s divide u_p)} --tau--> stop\\s
 : Specif of tcsp
```

- In *Boolean Enumeration Array* mode (BEA in short) all predicates are interpreted in a world of simple actions, without simultaneity product. The action world contains invertible signals denoted as $s!$ and $s?$ respectively, noninvertible actions denoted simply a. The hidden unobservable action is usually named τ, but can be redeclared as wished in the calculus description file (eg. i in PureLOTOS).

  The *enumeration* (the support of predicate interpretation) is made finite: we only consider the actions occurring in the expression, plus a finite number of extra signals and actions named ext_i, for other possible actions of the process variables in the expression.

```
@ parse x= local atom t in ((p[t/a] // {t} // q[t/b])\\t);
x : Process of tcsp

@ show eval x with sort {p, q} is signals none;
Specif of: local atom t in (p[t/a] //{t}//q[t/b] )\\t
   {q, p}{#<Cea u_p |a
                u_q |b
                res |i>}
           --i--> local atom t in (x_p[t/a] //{t}//x_q[t/b] )\\t
   {p}{#<Cea u_p |i b ext_1 ext_2
             res |i b ext_1 ext_2>}
        --u_p--> local atom t in (x_p[t/a] //{t}//q[t/b] )\\t
   {q}{#<Cea u_q |i a ext_1 ext_2
             res |i a ext_1 ext_2>}
        --u_q--> local atom t in (p[t/a] //{t}//x_q[t/b] )\\t
 : Specif of tcsp
```

- In *Composed Enumeration Array* mode (CEA) all predicates are interpreted in a world of compound actions, safe for the process variables, which are supposed to perform only atomic actions. Otherwise it resembles strongly the BEA mode, just with the addition of a simultaneity product in the resulting behaviours. In the case of finite specifications –where the *All* tactic suffices–, the complexity of resulting actions can be shown to be bounded.

```
@ parse x= ((a! * p) // (a? * q)) \ a!;
x : Process of meije

@ show eval x with  sort {p,q}  is signals none atoms any;
Specif of:  (a!*p//a?*q)\a!
   {q, p}{#<Cea |   u_p   u_q         res
                ------------------------
                |   tau   tau         tau
                |   tau ext_1       ext_1
                |   tau ext_2       ext_2
                | ext_1   tau       ext_1
                | ext_1 ext_1     ext_1^2
                | ext_1 ext_2 ext_1.ext_2
                | ext_2   tau       ext_2
                | ext_2 ext_1 ext_1.ext_2
                | ext_2 ext_2     ext_2^2>}
          --res-->
         (a!*x_p//a?*x_q)\a!  : Specif of meije
```

Even though the syntax of the predicates (see [MaDSiVe89]) was designed explicitly to represent the equivalent of rational sets in the commutative monoid of SCCS-MEIJE (also called semi-linear sets), this model was not implemented in ECRINS yet. All relevant problems (intersection, equality, complementation) would there be decidable, using linear homogeneous diophantine equations techniques ([GS64]), but with exponential complexity.

3.2.1 Bisimulation proving

ECRINS proves bisimulation in between open terms owing to the definitions of [DSi85]:

An equivalence relation $\Re$ over open process expressions is a **strong FH-bisimulation** *iff:*

$\forall p, q$ such that $p \,\Re\, q$,

$$\forall r \in \mathtt{Specif}(p) \text{ such that } \quad r = \frac{I.Pred}{p - p'}$$

$$\exists s \subseteq \mathtt{Specif}(q),\ s = \{s_k\}_k, \quad s_k = \frac{J.Pred_k}{q - q'_k}$$

$$\text{such that: } I = J \qquad \forall k,\ (p', q'_k) \in \Re \qquad \& \qquad Pred \subseteq \bigcup_k Pred_k$$

This means that each rule of one specif must correspond to one or more rules of the other specif. with same working free variables, equivalent resulting processes and a predicate "covered" by the union of the predicates of the corresponding rules.

As usual with bisimulations, the union of all FH-bisimulations is a FH-bisimulation. This largest FH-bisimulation is included in the largest bisimulation: completeness is not guaranteed, but differences are limited to some rare specific cases (see [DSi85]).

The user provides for a given relation (finite set of terms pairs) which is candidate for being a bisimulation. Actually we consider the substitutive, reflexive and symmetric closure of it. We do not close it by transitivity for decidability reasons. The system does not attempt to add more couple to this relation, not even interactively. This could be improved in the future. The system answer could be either a *Isbisim.* a *Fail* "don't know" or a *Notisbisim* answer. Cases of negative answer raise when two terms are found in a

couple such as one enjoys a rule with a provably non-empty predicate while there is no rule of the other term running on the same process variables, see [MaDSiVe89]

```
@ parse e1 = p+p == p;
e1 : Formula of meije

@ show prove e1;
|-- Isbisim {"p+p == p"} : Theorem of meije

@ parse e2 = {p+stop == p, p+q == q+p, (p+q)+r == p+(q+r),
              p//stop == p, p//q == q//p, (p//q)//r == p//(q//r),
              stop[phi] == stop, (u:p)[phi] == u<phi>:p[phi],
              a*stop == stop, a*b*p == (a.b)*p, tau*p == p,
              a*(b:p) == (a.b):a*p};
e2 : Formula of meije

@ prove e2;
: Theorem of meije
```

```
@ parse e3 = p//q == q;
e3 : Formula of meije

@ show prove e3;
Formula Disproved.
|-- Notisbisim {"p//q == q"} : Theorem of meije
```

For the time being, ECRINS treats only strong bisimulation. Research is ongoing towards observational bisimulation checking and abstract criteria (following [Bo85]). Several step evaluation have to be dealt with by incorporating resulting terms pairs in the relation.

4 Translation.

A lot of process calculi have been defined, but the translation between them was rarely done ([Hen82], [Mill87], [DSi85]). We present here an algebraic approach to (syntactic) translation of process calculi, and define the semantical validation of a simple translation. In algebraic structures, a translation is an *homomorphism* from source algebra to target algebra, defined inductively on operators.

4.1 Validity.

Informally, a translation will be said valid for a given equivalence $\sim$ iff semantics of the source program and its target translation coincide. Here validity will depend on finding an acceptable backward *semantic abstraction* $\mathcal{T}'$ such that the meaning of the translated program corresponds by $\mathcal{T}'$ to another semantic object, equivalent by $\sim$ to the meaning of the original program. This can be represented by the commutation diagram :

$\forall t \in S,\ \mathcal{T} : S \vdash T,$

$$\begin{array}{ccc} t & \xrightarrow{\text{syntactic translation } \mathcal{T}} & \mathcal{T}(t) \\ \downarrow & & \downarrow \\ [t]_S \sim \mathcal{T}'([\mathcal{T}(t)]_T) & \xrightarrow{\text{semantic abstraction } \mathcal{T}'} & [\mathcal{T}(t)]_T \end{array}$$

In our process calculi framework, the semantic meaning is defined by transition systems for closed terms and transition systems transformers for expressions. In the simple case where the source algebra of actions can be projected into a subset of the target algebra, the semantic abstraction can be the identity on these transition systems transformers. Moreover, we shall use here only strong bisimulation equivalences

Formally we say that a simple translation between process calculi $\mathcal{T}$ is *valid* (for the strong bisimulation) $\sim$ iff :

$$\forall t,\ t \sim \mathcal{T}(t)$$

4.2 Compositionality.

As we define the translation structurally on operators, we then prove the validity directly on each operator's case. The equivalence used must be a congruence to allow the induction principle: it is the case for the strong bisimulation.
Given :

1. Θ as the set of source operators and $Op \in \Theta$ an operator of arity I,
2. $\vec{p_i}$ as the vector of process variables of the operator Op,
3. $\mathcal{T}(\vec{p_i})$ as the translation of the process variable vector $\vec{p_i}$. It is a process variable vector too. Under the formal hypothesis, its elements must behave as source variables, i.e. are allowed to perform only source action (recall we assume here that source actions form a subset of target actions). In the sequel, we will note them simply $\vec{p_i}$ with a slight abuse of notation.

then the translation $\mathcal{T}$ will be *valid* (for the $\underset{FH}{\cong}$) iff :

$$\forall Op \in \Theta,\ \forall \vec{p_i},\ [Op(\vec{p_i})] \underset{FH}{\cong} [\mathcal{T}(Op)(\vec{p_i})]$$

5 A translation from PureLOTOS to MEIJE

We will show how we can prove the validity of a simple translation by ECRINS. We will define a syntactic translation from PureLOTOS to MEIJE and prove its validity for $\underset{FH}{\cong}$. This validation will be done by proving that the relation defined by the pairs $< Op, \mathcal{T}(Op) >$ is a bisimulation.

5.1 The translation

The calculus PLotosMeije is the union of the calculus MEIJE (zero, prefix, ticking, parallel, restriction, trigger, renaming, sum, see [Bo85] or [DSi85]) and PureLOTOS (stop, exit, hiding, rdv, disable, enable, choice, see [Ai86]), which is the pure synchronization kernel of LOTOS. A PureLOTOS action is an atom whose label is the corresponding LOTOS gate name.

We have already given in section 3.1 the description of the operators ticking (MEIJE), hiding and disable (PureLOTOS). A full definition of the union calculus PLotosMeije can be found in [DM88].

R. de Simone [DSi85] has proposed a constructive algorithm to translate a new operator in MEIJE, we have used here a similar construction pattern:

1. Exit : **exit** $\stackrel{\mathcal{T}}{\mapsto}$ d:stop

   The constant atomic action d represents the termination action δ of LOTOS.

2. Hiding : **hide** g **in** $p \stackrel{\mathcal{T}}{\mapsto} p[i/g]$.

   We rename a gate into the internal action to hide it.

3. Rendez-vous : $p|\{g\}|q \stackrel{\mathcal{T}}{\mapsto}$ local signals a, b in
   $\{(a! * p[b!.g/g]//a! * q[b!/g]//$let rec $\{h = a? : h + a??.b?? : h\}$ in $h)\backslash a!\backslash b!\backslash c!\}$.

   We control the two processes by sending only one occurrence of signal $a?$ and then force them to work independently. For the common gate g, we observe it with the signal $b!$ and send $b?$ only by pairs to force them to work together.

4. Enabling : $p >> q \stackrel{\mathcal{T}}{\mapsto}$ local signals $\{a, b, c\}$ in
   $(a! * p[c!/d]//b! => q//$ let rec $\{h = a? : h + a?.c? : b? : h\}$ in $h)\backslash a!\backslash b!\backslash c!$.

   We block the process q by the signal $b!$, which will be emitted only when p will terminates. The termination of p is observed through the signal $c!$.

   Notice we used here as safe the MEIJE renaming of the terminating d PureLOTOS gate signal.

5. Disabling : $p[>q \stackrel{\mathcal{T}}{\mapsto}$ local signals $\{a, b, c\}$ in
   $(a! * p[c!.d/d]//b! => q//$ let rec $\{l = b? : stop + a? : l + a?.c? : h\}$ and $\{h = a? : h + a?.c? : h\}$ in $l)\backslash a!\backslash b!\backslash c!$

   We have to distinguish here three behaviours: p performs an action, p terminates and q performs an action. The last two behaviours are driven respectively by the local signals $b!$ and $c!$:

   (a) if p terminates, then the clock l became the new clock h which blocks the process q,

   (b) if q performs an action the clock l will stop and then block p.

6. Choice : $p[]q \stackrel{\mathcal{T}}{\mapsto} p + q$.

5.2 Validation proof.

The validation proof is based on strong FH-bisimulation between the source operator and the target translation. The PureLOTOS variables do not perform any product of actions, so we can consider the CEA mode, to resolve the predicates associated to the specifications.

We cannot give here the whole validation, but we describe two interesting cases : hiding and disable. A complete validation proof can be found in [DM88].

```
@ parse t_hiding = p[i / a];
t_hiding : Process of plotosmeije

@ parse r = {t_hiding == p \\ a};
r : Relation of plotosmeije

@ show prove r;
-------------------------------------------------
Evaluation of left-hand-side gives :
Specif of: t_hiding
   {p}{#<Cea u_p |i a ext_1
             res |i i ext_1>}
        --res-->  x_p[i/a]

Evaluation of right-hand-side gives :
Specif of: p\\a
   {p}{#<Cea u_p |a
             res |i>}
        --res-->  x_p\\a
   {p}{#<Cea u_p |i ext_1
             res |i ext_1>}
        --res-->  x_p\\a

-------------------------------------------------
     Proof Succeeds
CEA |-- Isbisim {"t_hiding == (p\\a)"}
 : Theorem of plotosmeije
```

For the disabling case, we need to add new pairs to the initial relation; it is not as such a bisimulation. The new pairs are lemmas about the second or third step behaviours of the translated expression. They have been found needed during previous attempts of the proof, and have to be added by hand to the relation. The ECRINS system provides functions to extract the desired modified expressions after one or more evaluation so that you never need to type such terms by hand (see the use of *nextn* in the following example).

```
@ t_disabling = local signals {a, b, c} in
plotosmeije> (a! * p[ c!.d / d ] /// b! => q ///
plotosmeije>   let rec {l = b?:stop + a?:l + a?.c?:h and
plotosmeije>              h = a?:h + a?.c?:h} in l)
plotosmeije>   \a! \b! \c!;
t_disabling : Process of plotosmeije

@ set s = eval t_disabling;
s: Specif of plotosmeije

@ set t_disabling2 = nextn(2, s); set t_disabling3 = nextn(3, s);
t_disabling2 : Process of plotosmeije
t_disabling3 : Process of plotosmeije
```

```
@ parse r = {t_disabling == p [> q, t_disabling2 == x_p,
             t_disabling3 == x_q};
r : Relation of plotosmeije

@ show prove r;
-------------------------------------------------
 Processing Equation: "t_disabling == p[>q"
-------------------------------------------------
Evaluation of left-hand-side gives :
Specif of: t_disabling
   {p}{#<Cea u_p |i ext_1 ext_2
             res |i ext_1 ext_2>}
        --res-->
        local signals {a , b , c} in
          ((((a!*x_p[c!.d/d] ///b!=>q)///
               let rec {l = (b?:stop+a?:l)+a?.c?:h
                 and h = a?:h+a?.c?:h} in l)
              \a!)\b!)\c!
   {p}{#<Cea u_p |d
             res |d>}
        --res-->
        local signals {a , b , c} in
          ((((a!*x_p[c!.d/d] ///b!=>q)///
               let rec {l = (b?:stop+a?:l)+a?.c?:h
                 and h = a?:h+a?.c?:h} in h)
              \a!)\b!)\c!
   {q}{#<Cea u_q |i d ext_1 ext_2
             res |i d ext_1 ext_2>}
        --res-->   local signals {a , b , c} in
                   ((((a!*p[c!.d/d] ///x_q)///stop)\a!)\b!)\c!

Evaluation of right-hand-side gives :
Specif of: p[>q
   {p}{#<Cea u_p |i ext_1 ext_2
             res |i ext_1 ext_2>}
        --res-->  x_p[>q
   {p}{#<Cea u_p |d
             res |d>}
        --res-->  x_p
   {q}{#<Cea u_q |i d ext_1 ext_2
             res |i d ext_1 ext_2>}
        --res-->  x_q
-------------------------------------------------
     Proof Succeeds
-------------------------------------------------
 Processing Equation: "t_disabling2 == x_p"
-------------------------------------------------
Evaluation of left-hand-side gives :
```

```
Specif of: t_disabling2
   {x_p}{#<Cea u_x_p |i ext_1 ext_2
                res |i ext_1 ext_2>}
          --res-->
          local signals {a , b , c} in
            ((((a!*x_x_p[c!.d/d] ///b!=>q)///
                 let rec {l = (b?:stop+a?:l)+a?.c?:h
                     and h = a?:h+a?.c?:h} in h)
               \a!)\b!)\c!
   {x_p}{#<Cea u_x_p |d
                res |d>}
          --res-->
          local signals {a , b , c} in
            ((((a!*x_x_p[c!.d/d] ///b!=>q)///
                 let rec {l = (b?:stop+a?:l)+a?.c?:h
                    and h = a?:h+a?.c?:h} in h)
              \a!)\b!)\c!

Evaluation of right-hand-side gives :
Specif of: x_p
   {x_p}{#<Cea u_x_p |i d ext_1 ext_2
                res |i d ext_1 ext_2>}
          --res--> x_x_p

------------------------------------------------
     Proof Succeeds
------------------------------------------------
 Processing Equation: "t_disabling3 == x_q"
------------------------------------------------
Evaluation of left-hand-side gives :
Specif of: t_disabling3
   {x_q}{#<Cea u_x_q |i d ext_1 ext_2
                res |i d ext_1 ext_2>}
          --res-->
          local signals {a , b , c} in
            ((((a!*p[c!.d/d] ///x_x_q)///stop)\a!)\b!)\c!

Evaluation of right-hand-side gives :
Specif of: x_q
   {x_q}{#<Cea u_x_q |i d ext_1 ext_2
                res |i d ext_1 ext_2>}
          --res-->  x_x_q

------------------------------------------------
     Proof Succeeds
CEA |-- Isbisim {"t_disabling == p[>q",
                 "t_disabling2 == x_p",
                 "t_disabling3 == x_q"} : Theorem of plotosmeije
```

6 Conclusion

We have presented the ECRINS system, with an example automated validity proof for an intercalculi translation. Only strong bisimulation equivalences can be tested in the current version of ECRINS. It was all we needed for proving our translation from PureLOTOS to MEIJE correct, along with the possibility of renaming the termination action δ. Future developments will allow for correctness proofs of more complex translations, in between calculi with very different action mechanisms and atomic granularity. In such translations atomic actions of the source language may correspond to various sequences of actions in the implementation, requiring an abstraction mechanism over the behaviours. This will be achieved through the implementation of a broader class of bisimulations, running from observational equivalence, to equivalences parameterized by abstraction criteria (cf. [Bo85]).

The ECRINS tool can address a somehow different class of problems, studying properties of a given process calculus. As an example, we have obtained results on some operators of PureLOTOS: We have shown in [MaDSi87] how the *enable* operator can be simulated up to the weak bisimulation inside the PureLOTOS calculus:

```
let H = let rec {h0 = {sort_of_p}:h0 [] d_of_p:stop}
            in h0
    in
    ((p[d_of_p/d] // {sort_of_p, d_of_p} //H)
    // {d_of_p} //
    (d_of_p:q)) \\d_of_p
```

Moreover, if we consider only faithful processes, that is processes which are not allowed any τ action, orelse if processes are not allowed any behaviour after emitting a terminating action d, then the simulation happen to be valid with respect to strong bisimulation. Similarly, the *disable* operator can be simulated inside PureLOTOS whenever its process arguments are faithful.

Recursively defined processes may have infinite specifications that cannot be manipulated as such by the system; finite subsets of such specifications can be computed with evaluation tactics, but we have no mean to use our bisimulation algorithm in such cases. More research is still needed to obtain induction principles suitable for semi-automated proofs in such a framework.

Further developments include a more flexible management of calculi, allowing derived operators to be defined from a process expression, yielding a straightforward notion of subcalculi; the definition of a more general predicate evaluation mode; the implementation of a broader class of evaluation and bisimulation algorithms, allowing abstraction mechanisms based on the abstraction criterion concept.

References

[Ai86] G. Ailloux, "*Verification in Ecrins of Lotos Programs*", ESPRIT/SEDOS/C2/N45 (1986)

[BK85] Bergstra, Klop, "Algebra of Communicating Processes with Abstraction", *Theoretical Computer Science* 37, p77-121 (1985)

[Bo85] G. Boudol, "Notes on Algebraic Calculi of Processes", *Logics and Models of Concurrent Systems*, NATO ASI series F13, K.Apt ed. (1985)

[Br83] S. Brookes, "*A Model for Communicating Sequential Processes*", PhD Thesis, University of Oxford (1983)

[DM88] G. Doumenc, E. Madelaine, "*Une traduction de PLOTOS en MEIJE*", Rapport INRIA RR938 (1988)

[DSi85] R. De Simone, "Higher-Level Synchronising Devices in Meije-Sccs", *Theoretical Computer Science* 37, p245-267 (1985)

[GS64] S. Ginsburg, S. Spanier, "Bounded Algol-like languages", *Trans. Am. Math. Society* 113, p176-181 (1964)

[Hen82] M. Hennessy, W. Li, G. Plotkin, "*A First Attempt at Translating CSP Into CCS*", Second International Conf. on Distributed Computing Systems, Paris, 105-115 (1982)

[MaDSi87] E. Madelaine et R. de Simone, "*ECRINS, un laboratoire de preuve pour les calculs de processus*". Rapport INRIA RR672, (1987)

[MaDSiVe89] E. Madelaine, R. de Simone et D. Vergamini, "*ECRINS, A Proof Laboratory for Process Calculi, User Manual*", to appear INRIA (1989)

[Mi80] R. Milner, "*A Calculus for Communicating Systems*", Lectures Notes in Comput. Sci. 92 (1980)

[Mi83] R. Milner, "Calculi for Synchrony and Asynchrony", *Theoretical Computer Science* 25, p267-310 (1983)

[Mill87] M. Millington, "*Theories of translation correctness for concurrent programming languages*", Thesis, Univ. Edinburgh CST-46-87 (1987)

[Pl81] G. Plotkin, "*A Structural Approach to Operationnal Semantics*", Report Daimi FN-19, Comput. Sci. Dept., Aarhus Univ. (1981)

[Ve86] D. Vergamini, "*Verification by Means of Observational Equivalence on Automata*", Rapport INRIA RR501 (1986)

[Ve87] D. Vergamini, "*Vérification de réseaux d'automates finis par équivalences observationnelles: le systeème AUTO*, Thèse de doctorat Sciences, Université de Nice (1987)

[Ve89] D. Vergamini. "Verification of Distributed Systems: an Experiment", in *Formal Properties of Finite Automata and Applications*, LNCS 386, 1990

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique